Бодгин С.О., студент
Глазков А.В., студент
Зверева О.М., ст. преп.
Поршнев С.В., проф., д-р техн. наук

БАЗОВЫЕ ПРИНЦИПЫ ПОСТРОЕНИЯ ОБЪЕКТНО-ОРИЕНТИРОВАННЫХ ОПЕРАЦИОННЫХ СИСТЕМ

Современные операционные системы не могут справиться с огромным количеством проблем, возникших в последнее время. Среди таких проблем можно выделить защиту информации, зависимость программных комплексов от аппаратной платформы, сложности интерфейсов и, в частности, машиноориентированные файловые системы и т.д. В докладе поднимаются проблемы проектирования и программирования для современных ОС. В качестве решения предлагается пересмотреть основные принципы проектирования современных операционных систем. В качестве базовых принципов используется объектно-ориентированная парадигма программирования. Вводятся основные понятия и определения.

Описание объектно-ориентированной операционной системы

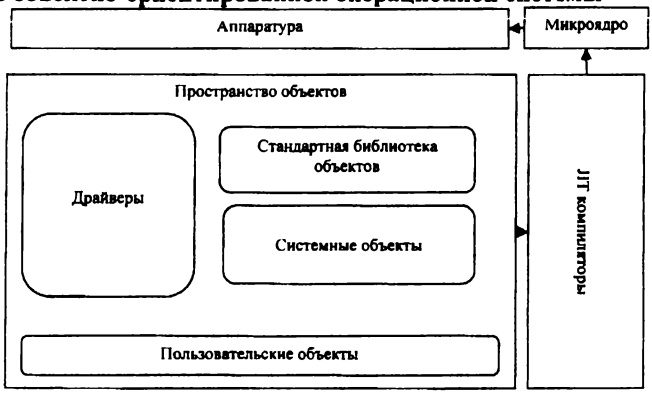

Рис. 1. Архитектура системы

На рис. 1 изображена архитектура системы. Основным элементом любой операционной системы является ядро. В ядре сосредотачивается вся базовая функциональность, необходимая для работы системы. Выделяется 2 основных типа ядер – монолитное ядро и микроядро. В первом случае ядро выполняет большое количество функций и является неделимым, во втором - ядро обладает минимальной функциональностью, а все другие задачи выполняют внешние, по отношению к ядру, программы-менеджеры. Наша система проектируется на основе микроядра с минимальным набором функций. Объекты в системе создаются на языках программирования высокого уровня и могут быть изменены в любое время. Just-In-Time или просто JIT-компиляторы создают исполняемый код процессора при первом запуске. И пока объект не изменился или кэш не был очищен, он выполняется из кэша без повторной компиляции.

На рис. 2 показано, каким образом происходит выполнение программного кода. Такая система позволяет обеспечивать объектно-ориентированность самой системы вне зависимости от языка, на котором пишет пользователь, и аппаратной части.

Архитектура поддерживает несколько языков программирования, несколько промежуточных байт-кодов и, следовательно, несколько JIT-компиляторов
(рис. 2). Например, имеет смысл в первых реализациях ориентироваться на Microsoft Framework .NET. Такая система может быть создана за более короткое время, т.к. есть возможность использовать уже разработанные open source решения. Среда разработки mono может служить неплохой средой для разработки объектной реализации. В качестве байт-кода будет выступать язык CIL, в качестве языка высокого уровня можно взять одну из разработок с открытым кодом. Это позволит получить реализацию системы за более короткий срок, а также поможет сформулировать основные требования к будущему собственному языку системы.

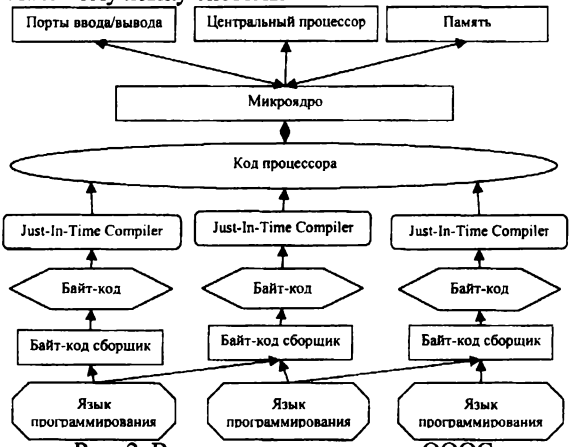

Рис. 2. Выполнение программ в OOOC

Объекты

Основным элементом объектно-ориентированной операционной системы, конечно, являются объекты. Объектная модель здесь соответствует всем принципам объектно-ориентированных систем. Весьма условно существует два вида объектов – статические и динамические. Первый тип объектов будем называть классами, а второй – объектами. Нужно заметить, что оба типа – это объекты, которые отличаются только по их использованию, и разделение введено только для большего понимания. Все принципы объектно-ориентированных систем выполняются. При реализации вопросов наследования возникает вопрос "версионности": как относится наследник к изменению кода родителя? Ведь любое изменение объекта можно провести в любой момент. В проектируемой системе принят следующий метод. Когда создается объект как наследник какого-либо другого объекта, либо, когда

существующему объекту добавляют родителя, создаётся два объекта: класс родителя (его текущее состояние) и от него два наследника, пустой наследник и сам наследник (рис. 3).

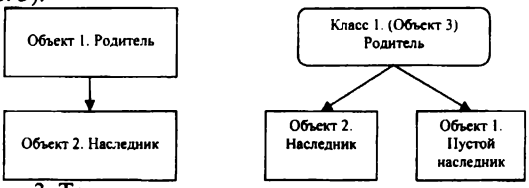

Рис. 3. Теоретическое и практическое наследование

Теперь при редактировании Объекта 1 (родителя) изменения не скажутся на наследнике. Созданный класс также можно изменить, но пользователь делает это на свой страх и риск и будет об этом предупрежден. Подробнее реализация этого алгоритма приведена далее в описании работы паттерна "FactoryMethod".

Операционная система, как бы представляет среду разработки в реальном времени. Мы создаём, удаляем и изменяем объекты. При выключении объекты меняют своё хранилище и переносятся, например, на жесткий диск. Таким образом, система работает непрерывно. После включения восстанавливается состояние, которое было до отключения. Для полного восстановления не требуется никаких специальных действий. В число системных объектов входят CallStack - этот объект хранит текущее состояние выполнения каждого метода – объекты, локальные переменные, строку выполнения и т.д. Если выполнение какого-то метода заканчивается, то мы удаляем верхний элемент по принципам стека и продолжаем выполнение там, где остановились. Все локальные переменные создаются в CallStack и удаляются автоматически после выхода на предыдущий уровень. Пример кода:

```
void foo( int arg1)
{
   ...
   int i = myCallStack.new Int; // Объект создаётся в стеке
   for( i=0;i<100;i++) print i;
   int j = mySystem.new Int; // Создаётся в общем хранилище
   ...
   mySystem.delete j; // Запрос к хранилищу на удаление объекта
} // Переменные стека удаляются без специального запроса
```

С помощью использования методов объектов можно создавать удобные WYSIWYG редакторы. Т.е., вызвав из метода одного объекта, либо из консоли, метод объекта-редактора и передав ему, в качестве параметра, нужный метод, можно редактировать наше окно или текст. На выходе, при запуске нужного метода, мы получим точную копию окна или текста, созданного в таком редакторе. Объекты занимаются своим отображением. Если пользователь хочет редактировать объект, то для этого он может воспользоваться любым редактором, который поддерживает этот класс данных (текст, музыка и т.д.)

Для большей универсальности в системе будет язык интерфейсов, который позволит стандартизовать все объекты. Если необходимо создать объект класса текст, то объект должен поддерживать соответствующий интерфейс. Например, для хранения графики придумано большое количество форматов, но свести можно к двум основным – растровые и векторные. Внутри объекта данные могут храниться в любом формате, но если объект поддерживает интерфейс BMPImage, то он должен позволить обратиться непосредственно к растру, как к обычному массиву.